RTI Data Distribution Service

The Real-Time Publish-Subscribe Middleware

Release Notes

Version 4.5d

The Global Leader in DDS

© 2011 Real-Time Innovations, Inc.
All rights reserved.
Printed in U.S.A. First printing.
Jan. 2011.

Trademarks

Real-Time Innovations and RTI are registered trademarks of Real-Time Innovations, Inc.
All other trademarks used in this document are the property of their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished under and subject to the RTI software license agreement. The software may be used or copied only under the terms of the license agreement.

Technical Support

Real-Time Innovations, Inc.
385 Moffett Park Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: https://support.rti.com/

Contents

1 System Requirements 2

1.1 Supported Operating Systems 2

1.2 Platforms Removed from RTI Data Distribution Service 4.5b 5

1.3 Disk and Memory Usage 5

1.4 Networking Support 5

2 Compatibility 15

2.1 Wire Protocol Compatibility 15

2.1.1 General Information on RTPS (All Releases) 15

2.1.2 Release-Specific Information for RTI Data Distribution Service 4.5 15

2.2 Code Compatibility 16

2.2.1 General Information (All Releases) 16

2.2.2 Release-Specific Information for RTI Data Distribution Service 4.5 17

2.3 Data-Format Compatibility and Extensibility 22

2.4 ODBC Database Compatibility 22

2.5 Build Compatibility 23

2.5.1 Windows CE C/C++ Build Compatibility 23

3 What's Fixed in 4.5d 23

3.1 Fixes Related to Reliability 24

3.1.1 Unnecessary Reliability Protocol Traffic from Reliable Reader if Writer Never Wrote Samples 24

3.1.2 Failure when Using a Reliable Asynchronous Publisher with Disabled Positive ACKs 24

3.1.3 Dynamic Send Window Stuck at Minimum Size 24

3.1.4 Piggyback Heartbeats Not Always Sent When Send Window Enabled 24

3.1.5 RELIABLE_READER_ACTIVITY_CHANGED StatusCondition Never Reset 25

3.1.6 Undocumented Behavior That Finite Send Window Caused KEEP_LAST DataWriters to Block and Potentilaly Fail on Write 25

3.2 Fixes Related to Durability 25
3.2.1 Lost Samples in Late-Joining Reader when using Keyed Topic and KEEP_LAST History 25
3.2.2 Slow Restore Operation when using Keyed Topics and dds.data_writer.history.odbc_plugin.in_memory_state Set to 1 26
3.3 Fixes Related to QueryConditions and ReadConditions 26
3.3.1 Loss of StateMask Values During QueryCondition's set_query_parameters() 26
3.4 Fixes Related to rtiddsgen 26
3.4.1 Error Parsing XML Unions with Enums as Discriminators—C API Only 26
3.4.2 Some Constants Generated with -convertToXML were not Language-Neutral 27
3.4.3 Use of Keyword 'any' in IDL Resulted in Code that Did Not Compile 27
3.4.4 USER_QOS_PROFILE.XML Not Generated for Mac OS X Platform 27
3.4.5 Error when Running rtiddsgen with Java Option and XSD in Another Directory 27
3.4.6 Incorrect Code Generated if XSD Enumerations had Non-Ordinal Annotations 28
3.4.7 Incorrect XML Generated When Using Some Preprocessors 28
3.4.8 Error from rtiddsgen if IDL File Contained Unicode non-ASCII Characters 28
3.4.9 Issues when using rtiddsgen -D Incorrectly 28
3.4.10 Wrong Visual Studio Properties for Hello_idl Example and Generated Examples 29
3.4.11 Error Parsing XML Typedefs with Non-Literal Dimensions 29
3.4.12 Incorrect Const Enums when Using rtiddsgen -package option 29
3.4.13 Error Running rtiddsgen for Some Linux Distributions 30
3.4.14 Incorrect Type Code Name when Using -package 30
3.4.15 Incorrect Case Values for Unions with Enum Discriminators 30
3.4.16 Compiler Error if IDL Union had Field Named 'result' or 'value'—Java API Only 31
3.4.17 Inconsistent Header Files Created When Generating Code for C or C++ 31
3.5 Fixes Related to TCP Transport 31
3.5.1 TCP Transport Plugin Aborted During Participant Restart 31
3.5.2 Possible Crash after Error in TCP Transport Protocol 31
3.5.3 Segmentation Fault if TCP Transport Failed to Create Socket 32
3.6 Fixes Related to QoS and Status 32
3.6.1 Incorrect uncommitted_sample_count Status Caused '!precondition' Exception or Crash 32
3.6.2 Use of cpu_list in Thread Settings May Have Caused Memory Corruption 32
3.6.3 Some Participant Properties not Propagated, Serialization Error Reported 33
3.6.4 Fields in DataWriterProtocolStatus Not Properly Filled In 33

3.6.5 Partition Matching Errors after Partition is Changed 33
3.7 Fixes Related to Dynamic Data in Java API 33
3.7.1 DynamicData.get_string() Failed When Using Member ID—Java API Only 33
3.7.2 Setting of Unsigned Types Not Supported with Dynamic Data API—Java API Only 34
3.8 Fixes Related to ContentFilteredTopics 34
3.8.1 Best-Effort Writer-Side Content-Filtered Samples Reported As Lost 34
3.8.2 Reliable Content-Filtering Writer May Have Incorrectly Inactivated Reader 35
3.8.3 No Warning if contentfilter_property_max_length was Too Small 35
3.8.4 Improper Serialization of content_filter_property for SubscriptionBuiltinTopicData 35
3.8.5 Failure When Immediately Resending Writer-side Content-Filtered Sample 35
3.9 Other Fixes 36
3.9.1 Dispose Samples not Generated for Remote Participant's Contained Entities After Ungraceful Termination 36
3.9.2 Error when Using Separate Threads to Enable Participant and Create DataReader at the Same Time 36
3.9.3 Wrong Behavior for Null Listener with Non-Null Mask—Java API Only 36
3.9.4 DATA_ON_READERS Status not Reset on Call to take() 36
3.9.5 Problem Redirecting Logging to File During DomainParticipantFactory Deletion. 36
3.9.6 Memory In Use After DDS Application Exited—C++ API Only 37
3.9.7 Reader Blocked Indefinitely when Writers Sent Large Data 37
3.9.8 Multichannel DataWriter did not Use Multicast if DataReader had not Defined a Content Filter 37
3.9.9 Possible Crash when Configuring DataWriter to Use Asynchronous Publisher 37
3.9.10 Error Message if Type Code Max. Serialized Length > 65532 37
3.9.11 DataReaderProtocolStatus.first_unacknowledged_sample_subscription_handle not Always Updated 37
3.9.12 Crash after Setting Logging Delegate while Monitoring Enabled—.Net API Only. 38
3.9.13 Visual Studio 6.0 on Windows 2000 was not Supported 38
3.9.14 Sequence-number in Statuses Displayed as Negative Values for Best-effort DataWriters 38
3.9.15 Unnecessary Exception When Monitoring DataWriter with Transient or Persistent Durability 38

4 Known Issues 39
4.1 Estimate of Message Overhead For Large Data May Be Exceeded 39
4.2 Writer-side Filtering May Cause a Deadline to be Missed 39
4.3 Disabled Interfaces on Windows Systems 39
4.4 Wrong Error Code After Timeout on write() from Asynchronous Publisher 39
4.5 Incorrect Content Filtering for Valuetypes and Sparse Types 39
4.6 Multi-dimensional Arrays of Sequences Not Supported when Generating .NET Code 40
4.7 License File may not be Found if NDDSHOME Set with Quotation Marks 40
4.8 Errors when Using Monitoring Library and Long Names/Values in PropertyQosPolicy 40
4.9 Changes Required when Using More than 5 Content-Filter Parameters 41
4.10 Issues with Dynamic Data 41

Release Notes

This document includes the following sections:

- System Requirements (Section 1)
- Compatibility (Section 2)
- What's Fixed in 4.5d (Section 3)
- Known Issues (Section 4)

For an overview of new features, please see the *What's New* document (RTI_DDS_WhatsNew.pdf).

For more information, visit the RTI Knowledge Base, accessible from https://support.rti.com/, to see sample code, general information on *RTI® Data Distribution Service*, performance information, troubleshooting tips, and technical details. By its very nature, the knowledge base is continuously evolving and improving. We hope that you will find it helpful. If there are questions that you would like to see addressed or comments you would like to share, please send e-mail to **support@rti.com**. We can only guarantee a response to customers with a current maintenance contract or subscription. You can purchase a maintenance contract or subscription by contacting your local RTI representative (see http://www.rti.com/company/contact.html), sending an e-mail request to **sales@rti.com**, or calling +1 (408) 990-7400.

1 System Requirements

1.1 Supported Operating Systems

RTI Data Distribution Service requires a multi-threaded operating system. This section describes the host and target systems supported by *RTI Data Distribution Service*.

In this context, a *host* is the computer on which you will be developing a *RTI Data Distribution Service* application. A *target* is the computer on which the completed application will run. A host installation provides the code generation tool (*rtiddsgen*), examples and documentation, as well as the header files required to build an *RTI Data Distribution Service* application for any architecture. You will also need a target installation, which provides the libraries required to build an *RTI Data Distribution Service* application for that particular target architecture.

Table 1.0 lists the platforms available with *RTI Data Distribution Service* 4.5d.

Table 1.0 **Platforms Available with Release 4.5d**

Platform	Operating System	Reference
AIX®	AIX 5.3	Table 1.1 on page 6
INTEGRITY®	INTEGRITY 5.0	Table 1.2 on page 6
Linux® (Cell BE™)	Fedora™ 9	Table 1.3 on page 6
Linux (Intel®)	CentOS 5.4, 5.5 Fedora 10 Red Hat® Enterprise Linux 3.0, 4.0, 5.0-5.2, 5.4, 5.5 Red Hat Enterprise Linux 5.2 with Real-Time Extensions Red Hat Linux 8.0 and 9.0 SuSE® Linux Enterprise Server 10.1 (2.6 kernel) Ubuntu® Server 10.04 (2.6 kernel)	Table 1.4 on page 7
Linux (PowerPC®)	SELinux (2.6.27.14) Yellow Dog™ Linux 4.0	Table 1.5 on page 8
LynxOS®	LynxOS 4.0, LynxOS 4.2, LynxOS 5.0, LynxOS-SE 3.0,	Table 1.6 on page 9
Mac OS®	Mac OS X	Table 1.7 on page 9
Solaris™	Solaris 2.9 and 2.10	Table 1.8 on page 9
VxWorks®	VxWorks 5.4.2, 5.5.1, 6.0 - 6.7	Table 1.9 on page 10

Table 1.0 **Platforms Available with Release 4.5d**

Platform	Operating System	Reference
Windows®	Windows 7 Windows 2000 with service pack 2 or higher Windows 2003 and Windows 2003 x64 Edition Windows CE 6.0 Windows Vista® Windows Server 2008 R2 Windows XP Professional and Windows XP Professional x64 Edition	Table 1.10 on page 12

Visual Studio® 2005 — Service Pack 1 Requirement

❑ You must have Visual Studio 2005 Service Pack 1 or the Microsoft Visual C++ 2005 SP1 Redistribution Package installed on the machine where you are *running* an application built with the release libraries of the following RTI architecture packages:

- x64Win64VS2005 built with dynamic libraries
- i86Win32VS2005 built with dynamic libraries
- i86Win32jdk, x64Win64jdk, i86Win32dotnet2.0, and x64Win64dotnet2.0

The Microsoft Visual C++ 2005 Service Pack 1 Redistribution Package can be downloaded from RTI's Customer Portal[1], or obtained from the following Microsoft website:

- For x86 architectures:
  http://www.microsoft.com/downloads/details.aspx?familyid=200B2FD9-AE1A-4A14-984D-389C36F85647&displaylang=en
- For x64 architectures:
  http://www.microsoft.com/downloads/details.aspx?familyID=EB4EBE2D-33C0-4A47-9DD4-B9A6D7BD44DA&displaylang=en

To run an application built with *debug* libraries of the above RTI architecture packages, you must have Visual Studio 2005 Service Pack 1 installed.

1. On the portal, select the Downloads page. The Redistribution Package is in the section labeled, "Windows Target Libraries for RTI Data Distribution Service."

Visual Studio 2008 - Service Pack 1 Requirement

- You must have Visual Studio 2008 Service Pack 1 or the Microsoft Visual C++ 2008 SP1 Redistribution Package installed on the machine where you are *running* an application built with the release libraries of the following RTI architecture packages:
  - x64Win64VS2008 built with dynamic libraries
  - i86Win32VS2008 built with dynamic libraries

  To run an application built with *debug* libraries of the above RTI architecture packages, you must have Visual Studio 2008 Service Pack 1 installed.

  The Microsoft Visual C++ 2008 Service Pack 1 Redistribution Package can be downloaded from RTI's Customer Portal[1], or obtained from the following Microsoft website:
  - For x86 architectures:

    http://www.microsoft.com/downloads/details.aspx?familyid=A5C84275-3B97-4AB7-A40D-3802B2AF5FC2&displaylang=en
  - For x64 architectures:

    http://www.microsoft.com/downloads/details.aspx?FamilyID=ba9257ca-337f-4b40-8c14-157cfdffee4e&displaylang=en

Note: Additional platforms not listed in this document may be supported through special development and maintenance agreements. Contact your RTI sales representative for details.

The following tables provide additional details. See the *RTI Data Distribution Service User's Manual* and *Platform Notes* for more information on compilers and linkers.

- Table 1.1, "AIX Platforms," on page 1-6
- Table 1.2, "INTEGRITY Platforms," on page 1-6
- Table 1.3, "Linux Platforms on Cell BE CPUs," on page 1-6
- Table 1.4, "Linux Platforms on Intel and AMD CPUs," on page 1-7
- Table 1.5, "Linux Platforms on PowerPC CPUs," on page 1-8
- Table 1.6, "LynxOS Platforms," on page 1-9
- Table 1.7, "Mac OS Platforms," on page 1-9
- Table 1.8, "Solaris Platforms," on page 1-9
- Table 1.9, "VxWorks Platforms," on page 1-10
- Table 1.10, "Windows Architectures," on page 1-12

1.2 Platforms Removed from RTI Data Distribution Service 4.5b

Beginning with 4.5b, these platforms are no longer supported:

- ❏ Fedora 8 (ppc64Linux2.6gcc4.1.2)
- ❏ J2SE 1.4 for most platforms
- ❏ LynxOS 4.0 on i86 CPUs for gcc 2.95.3 (i86Lynx4.0.0gcc2.95.3)
- ❏ LynxOS 4.2 on PPC 604 and PPC 7XX for gcc 3.2.2 (ppc750Lynx4.2.0gcc3.2.2)
- ❏ QNX Neutrino 6.3.0
- ❏ Solaris 2.8
- ❏ Solaris 2.9 on UltraSPARC for CC 5.3 (sparcSol2.9cc5.3)

1.3 Disk and Memory Usage

Disk usage for a typical host-only installation is approximately 250 MB. Each additional architecture (host or target) requires an additional 75 MB.

We recommend that you have at least 256 MB RAM installed on your host development system. The target requirements are significantly smaller and they depend on the complexity of your application and hardware architecture.

1.4 Networking Support

RTI Data Distribution Service includes full support for pluggable transports. *RTI Data Distribution Service* applications can run over various communication media, such as UDP/IP over Ethernet, and local inter-process shared memory—provided the correct "transport plug-ins" for the media are installed.

By default, *RTI Data Distribution Service* uses the UDP/IPv4 and shared-memory transport plug-ins. The shared memory transport is not supported for VxWorks 5.4 and 5.5.

A built-in IPv6 transport is also available (disabled by default) for these platforms:

- ❏ Linux/Fedora: all platforms except Red Hat Linux 8.0 and 9.0, and SELinux
- ❏ Solaris: all platforms
- ❏ Windows: all platforms except Windows CE and Visual C 6.0/7.0

A TCP transport is also available (but is not a *built-in* transport) for the following platforms:

- ❏ RedHat Enterprise Linux 4.0, 5.0-5.2, 5.4, 5.5; CentOS 5.4, 5.4; and Ubuntu 10.04
- ❏ Windows with Visual Studio 2005, 2008, and 2010.

Supported architectures appear on the following pages, followed by Compatibility (Section 2).

Table 1.1 **AIX Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
AIX 5.3	POWER5 (32-bit mode)	IBM XLC for AIX v9.0	p5AIX5.3xlc9.0
		IBM Java 1.6	p5AIX5.3xlc9.0jdk
	POWER5 (64-bit mode)	IBM XLC for AIX v9.0	64p5AIX5.3xlc9.0
		IBM Java 1.6	64p5AIX5.3xlc9.0jdk

Table 1.2 **INTEGRITY Platforms**

Operating System	CPU	IP Stack[1]	RTI Architecture Abbreviation
INTEGRITY 5.0.7	PPC 74XX	InterNiche (GHnet1) TCP/IP stack	ppc7400Inty5.0.7.mvme5100-7400 [2]
		Interpeak TCP/IP stack with multicast	ppc7400Inty5.0.7.mvme5100-7400-ipk
INTEGRITY 5.0.8	PPC 74XX	InterNiche (GHnet1) TCP/IP stack	ppc7400Inty5.0.7.mvme5100-7400[2]
INTEGRITY 5.0.9, 5.0.10	PPC 74XX	GHnet2 TCP/IP stack	ppc7400Inty5.0.9.mvme5100-7400-ghnet2

1. For additional supported transports, see the online documentation or contact support@rti.com.
2. INTEGRITY 5.0.7 and 5.0.8 share the same architecture (ppc7400Inty5.0.7.mvme5100-7400).

Table 1.3 **Linux Platforms on Cell BE CPUs**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Fedora 9 (2.6 kernel)	Cell BE	ppu 4.1.1	cell64Linux2.6ppu4.1.1
		IBM Java JDK 1.5	cell64Linux2.6ppu4.1.1jdk

Table 1.4 **Linux Platforms on Intel and AMD CPUs**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
CentOS 5.4, 5.5 (2.6 kernel)	Pentium class	gcc 4.1.2	i86Linux2.6gcc4.1.2
		Sun Java Platform Standard Edition JDK 1.6	i86Linux2.6gcc4.1.2jdk
	x64	gcc 4.1.2	x64Linux2.6gcc4.1.2
		Sun Java Platform Standard Edition JDK 1.6	x64Linux2.6gcc4.1.2jdk
Fedora 10 (2.6 kernel)	x64	gcc 4.3.2	x64Linux2.6gcc4.3.2
		IBM Java JDK 1.6	x64Linux2.6gcc4.3.2jdk
Red Hat Linux 8.0 (2.4 kernel)	Pentium class	gcc 3.2	i86Linux2.4gcc3.2
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.4gcc3.2jdk
Red Hat Linux 9.0 (2.4 kernel)	Pentium class	gcc 3.2.2	i86Linux2.4gcc3.2.2[1]
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.4gcc3.2.2jdk
Red Hat Enterprise Linux 3.0 (2.4 kernel)	Pentium class	gcc 3.2.2	i86Linux2.4gcc3.2.2[1]
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.4gcc3.2.2jdk
	AMD64	gcc 3.2.3	x64Linux2.4gcc3.2.3
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	x64Linux2.4gcc3.2.3jdk
Red Hat Enterprise Linux 4.0 (2.6 kernel)	Pentium class	gcc 3.4.3	i86Linux2.6gcc3.4.3
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.6gcc3.4.3jdk
	x86_64 and AMD64	gcc 3.4.5	x64Linux2.6gcc3.4.5
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	x64Linux2.6gcc3.4.5jdk
Red Hat Enterprise Linux 5.0 (2.6 kernel)	Pentium class	gcc 4.1.1	i86Linux2.6gcc4.1.1
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.6gcc4.1.1jdk
	x86_64 and AMD64	gcc 4.1.1	x64Linux2.6gcc4.1.1
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	x64Linux2.6gcc4.1.1jdk

Table 1.4 **Linux Platforms on Intel and AMD CPUs**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Red Hat Enterprise Linux 5.1, 5.2, 5.4, 5.5 (2.6 kernel)	Pentium class	gcc 4.1.2	i86Linux2.6gcc4.1.2
		Sun Java Platform Standard Edition JDK 1.6	i86Linux2.6gcc4.1.2jdk
	x86_64 and AMD64	gcc 4.1.2	x64Linux2.6gcc4.1.2
		Sun Java Platform Standard Edition JDK 1.6	x64Linux2.6gcc4.1.2jdk
Red Hat Enterprise Linux 5.2 with Real-Time Extensions (2.6 kernel)	Pentium class	gcc 4.1.2	i86Linux2.6gcc4.1.2
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.6gcc4.1.2jdk
SUSE Linux Enterprise Server 10.1 (2.6 kernel)	Pentium class	gcc 4.1.0	i86Suse10.1gcc4.1.0
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Suse10.1gcc4.1.0jdk
	AMD64	gcc 4.1.0	x64Suse10.1gcc4.1.0
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	x64Suse10.1gcc4.1.0jdk
Ubuntu Server 10.04 (2.6 kernel)	Pentium class	gcc 4.4.3	i86Linux2.6gcc4.4.3
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.6gcc4.4.3jdk
	x64	gcc 4.4.3	x64Linux2.6gcc4.4.3
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	x64Linux2.6gcc4.4.3jdk

1. Red Hat Linux 9.0 and Red Hat Enterprise Linux 3.0 share the same RTI architecture.

Table 1.5 **Linux Platforms on PowerPC CPUs**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
SELinux (2.6.27.14)	PowerPC440EP	gcc 4.3.3 with GNU libc 2.9	ppc4xxFPLinux2.6gcc4.3.3
Yellow Dog® Linux 4.0 (2.6 kernel) (target only)	PPC 74xx (such as 7410)	gcc 3.3.3	ppc7400Linux2.6gcc3.3.3

Table 1.6 **LynxOS Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
LynxOS 4.0	Pentium class	gcc 3.2.2	i86Lynx4.0.0gcc3.2.2
		Sun Java Platform Standard Edition JDK 1.4	i86Lynx4.0.0gcc3.2.2jdk
	PPC 74xx (such as 7410)	gcc 3.2.2	ppc7400Lynx4.0.0gcc3.2.2
		Sun Java Platform Standard Edition JDK 1.4	ppc7400Lynx4.0.0gcc3.2.2jdk
	PPC 604 PPC 7XX (such as 750)	gcc 3.2.2	ppc750Lynx4.0.0gcc3.2.2
		Sun Java Platform Standard Edition JDK 1.4	ppc750Lynx4.0.0gcc3.2.2jdk
LynxOS 4.2	Pentium class	gcc 3.2.2	i86Lynx4.2.0gcc3.2.2
	PPC 74xx (such as 7410)	gcc 3.2.2	ppc7400Lynx4.2.0gcc3.2.2
LynxOS 5.0	PPC 74xx (such as 7410)	gcc 3.4.3	ppc7400Lynx5.0.0gcc3.4.3
		Sun Java Platform Standard Edition JDK 1.4	ppc7400Lynx5.0.0gcc3.4.3jdk
LynxOS-SE 3.0	Pentium class	gcc 3.4.3	i86LynxOS_SE3.0.0gcc3.4.3

Table 1.7 **Mac OS Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Mac OS X	x64	gcc 4.2.1	x64Darwin10gcc4.2.1
		Java SE 1.6 for Mac OS	x64Darwin10gcc4.2.1jdk

Table 1.8 **Solaris Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Solaris 2.9	Pentium class	gcc 3.3.2	i86Sol2.9gcc3.3.2
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Sol2.9jdk
	UltraSPARC	CC 5.4 (Forte Dev 7, Sun One Studio 7)	sparcSol2.9cc5.4
		gcc 3.2	sparcSol2.9gcc3.2
		gcc 3.3	sparcSol2.9gcc3.3
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	sparcSol2.9jdk

Table 1.8 **Solaris Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Solaris 2.10	AMD64	gcc 3.4.3	x64Sol2.10gcc3.4.3
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Sol2.10jdk
	Pentium class	gcc 3.4.4	i86Sol2.10gcc3.4.4
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Sol2.10jdk
	UltraSPARC	gcc3.4.2	sparcSol2.10gcc3.4.2
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	sparcSol2.10jdk
	UltraS-PARC (with native 64-bit support)	cc 5.8	sparc64Sol2.10cc5.8
		gcc3.4.2	sparc64Sol2.10gcc3.4.2
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	sparc64Sol2.10jdk

Table 1.9 **VxWorks Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
VxWorks 5.4.2	ppc604, ppc750, ppc7400	gcc 2.96	ppc604Vx5.4gcc
VxWorks 5.5.1	Pentium	gcc 2.96	pentiumVx5.5gcc
	ppc405		ppc405Vx5.5gcc
	ppc604, ppc750, ppc7400		ppc604Vx5.5gcc
	ppc603		ppc603Vx5.5gcc
VxWorks 6.0, 6.1, 6.2	Pentium	gcc 3.3.2	For kernel modules: pentiumVx6.0gcc3.3.2 For Real Time Processes: pentiumVx6.0gcc3.3.2_rtp
	Any Wind River PPC32 CPU with floating point hardware		For kernel modules: ppc604Vx6.0gcc3.3.2 For Real Time Processes: ppc604Vx6.0gcc3.3.2_rtp

Table 1.9 **VxWorks Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
VxWorks 6.3, 6.4	Pentium	gcc 3.4.4	For kernel modules: pentiumVx6.3gcc3.4.4 For Real Time Processes: pentiumVx6.3gcc3.4.4_rtp
	Any Wind River PPC32 CPU with floating point hardware		For kernel modules: ppc604Vx6.3gcc3.4.4 For Real Time Processes: ppc604Vx6.3gcc3.4.4_rtp
VxWorks 6.5	Pentium	gcc 3.4.4	For kernel modules: pentiumVx6.5gcc3.4.4 For Real Time Processes: pentiumVx6.5gcc3.4.4_rtp
	Any Wind River PPC32 CPU with floating point hardware		For kernel modules: ppc604Vx6.5gcc3.4.4 For Real Time Processes: ppc604Vx6.5gcc3.4.4_rtp
VxWorks 6.6	Pentium	gcc 4.1.2	For Kernel Modules: pentiumVx6.6gcc4.1.2 For Real Time Processes: pentiumVx6.6gcc4.1.2_rtp
	Any Wind River PPC32 CPU with floating point hardware	gcc 4.1.2	For Kernel Modules: ppc604Vx6.6gcc4.1.2 For Real Time Processes: ppc604Vx6.6gcc4.1.2_rtp
	ppc405	gcc 4.1.2	For Kernel Modules: ppc405Vx6.6gcc4.1.2 For Real Time Processes: ppc405Vx6.6gcc4.1.2_rtp

Table 1.9 **VxWorks Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
VxWorks 6.7	Pentium	gcc 4.1.2	For Kernel Modules: pentiumVx6.7gcc4.1.2 For Real Time Processes: pentiumVx6.7gcc4.1.2_rtp
	any Wind River PPC32 CPU with floating point hardware	gcc 4.1.2	For Kernel Modules: ppc604Vx6.7gcc4.1.2 For Real Time Processes on a non-SMP system: ppc604Vx6.7gcc4.1.2_rtp For Real Time Processes on an SMP system: ppc604Vx6.7gcc4.1.2_smp
	ppc405[1]	gcc 4.1.2	For Kernel Modules: ppc405Vx6.6gcc4.1.2 For Real Time Processes: ppc405Vx6.6gcc4.1.2_rtp

1. For ppc405, the architecture string is the same for VxWorks 6.6 and 6.7.

Table 1.10 **Windows Architectures**

Operating System	CPU	Compiler or Software Development Kit [1] [2]	RTI Architecture
Windows 7	x86	Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	i86Win32dotnet2.0
		Visual Studio 2010	i86Win32VS2010
		Visual Studio 2010 (C++, C# 8.0 or 9.0)	i86Win32dotnet4.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk
Windows 7 x64 Edition	x64	Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	x64Win64dotnet2.0
		Visual Studio 2010	x64Win64VS2010
		Visual Studio 2010 (C++, C# 8.0 or 9.0)	x64Win64dotnet4.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk

Table 1.10 **Windows Architectures**

Operating System	CPU	Compiler or Software Development Kit [1] [2]	RTI Architecture
Windows 2000	x86	Visual C 6.0	i86Win32VC60
		Visual C 7.0	i86Win32VC70
		Visual Studio 2003	i86Win32VS2003
		Visual Studio 2005 SP1	i86Win32VS2005
		Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	i86Win32dotnet2.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk
Windows 2003	x86	Visual C 6.0	i86Win32VC60
		Visual C 7.0	i86Win32VC70
		Visual Studio 2003	i86Win32VS2003
		Visual Studio 2005 SP 1	i86Win32VS2005
		Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	i86Win32dotnet2.0
		Visual Studio 2008 SP1	i86Win32VS2008
		Visual Studio 2008 SP 1 (C++, C# 8.0 or 9.0)	i86Win32dotnet2.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk
Windows 2003 x64 Edition	x64	Visual Studio 2005 SP 1	x64Win64VS2005
		Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	x64Win64dotnet2.0
		Visual Studio 2008 SP 1	x64Win64VS2008
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk
Windows CE 6.0 (target only)[3] [4] [5]	armv4	Visual Studio 2005 SP 1 (C++ 8.0)	armv4WinCE6.0VS2005
	x86	Visual Studio 2005 SP 1 (C++ 8.0)	i86WinCE6.0VS2005
Windows Server 2008 R2 x64 Edition	x64	Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	x64Win64dotnet2.0
		Visual Studio 2010	x64Win64VS2010
		Visual Studio 2010 (C++, C# 8.0 or 9.0)	x64Win64dotnet4.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk
Windows Vista	x86	Visual Studio 2005 SP 1	i86Win32VS2005
		Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	i86Win32dotnet2.0
		Visual Studio 2008 SP 1	i86Win32VS2008
		Visual Studio 2008 SP 1 (C++, C# 8.0 or 9.0)	i86Win32dotnet2.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk

Table 1.10 **Windows Architectures**

Operating System	CPU	Compiler or Software Development Kit [1] [2]	RTI Architecture
Windows Vista 64-bit Edition	x64	Visual Studio 2005 SP 1	x64Win64VS2005
		Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	x64Win64dotnet2.0
		Visual Studio 2008 SP1	x64Win64VS2008
		Visual Studio 2008 SP 1 (C++, C# 8.0 or 9.0)	x64Win32dotnet2.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk
Windows XP Professional [6]	x86	Visual C 6.0	i86Win32VC60
		Visual C 7.0	i86Win32VC70
		Visual Studio 2003	i86Win32VS2003
		Visual Studio 2005 SP 1	i86Win32VS2005
		Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	i86Win32dotnet2.0
		Visual Studio 2008 SP 1	i86Win32VS2008
		Visual Studio 2008 SP 1 (C++, C# 8.0 or 9.0)	i86Win32dotnet2.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk
Windows XP Professional x64 Edition	x64	Visual Studio 2005 SP 1	x64Win64VS2005
		Visual Studio 2005 SP 1 (C++, C# 8.0 or 9.0)	x64Win64dotnet2.0
		Visual Studio 2008 SP 1	x64Win64VS2008
		Visual Studio 2008 SP 1 (C++, C# 8.0 or 9.0)	x64Win32dotnet2.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk

1. On Windows XP: If you are using JDK 5.0 and want to use Intel's HyperThreading technology, use JDK 5.0 Update 6 (build 1.5.0_06), which includes fixes to JNI and HyperThreading. (If you must use Update 5 (build 1.5.0_05), you should disable HyperThreading.)
2. The RTI .NET assemblies are supported for both the C++/CLI and C# languages. The type support code generated by rtiddsgen is in C++/CLI; compiling the generated type support code requires Microsoft Visual C++. Calling the assembly from C# requires Microsoft Visual C#.
3. The Windows CE network stack does not support the IP_TOS socket option.
4. The Windows CE device must be connected directly to the network, not through a Windows PC using ActiveSync. RTI Data Distribution Service does not support Windows CE when used with ActiveSync
5. On Windows CE systems, add registry values under HKEY_LOCAL_MACHINE\Software\RTI for the environment variables that RTI Data Distribution Service looks up. For example: To set NDDS_DISCOVERY_PEERS to 127.0.0.1, under "HKEY_LOCAL_MACHINE\Software\RTI", add a string value with the name "NDDS_DISCOVERY_PEERS" and data "127.0.0.1".
6. Windows XP does not support IP_TOS unless registry changes are made. See http://support.microsoft.com/kb/248611, http://www.microsoft.com/technet/technetmag/issues/2007/02/CableGuy/default.aspx.

2 Compatibility

RTI strives to provide a seamless upgrade path when the product is updated. When upgrading to a new version of *RTI Data Distribution Service*, there are five components to consider:

- ❏ Wire Protocol Compatibility (Section 2.1)
- ❏ Code Compatibility (Section 2.2)
- ❏ Data-Format Compatibility and Extensibility (Section 2.3)
- ❏ ODBC Database Compatibility (Section 2.4)
- ❏ Build Compatibility (Section 2.5)

2.1 Wire Protocol Compatibility

2.1.1 General Information on RTPS (All Releases)

RTI Data Distribution Service communicates over the wire using a formal Real-time Publish-Subscribe (RTPS) protocol.

RTPS has been developed from the ground up with performance, interoperability and extensibility in mind. The RTPS protocol is an international standard managed by the OMG. The RTPS protocol has built-in extensibility mechanisms that enable new revisions to introduce new message types, extend the existing messages, or extend the Quality of Service settings in the product—without breaking interoperability.

RTPS 1.0 was introduced in 2001. The current version is 2.1. RTI plans to maintain interoperability between middleware versions based on RTPS 2.x.

2.1.2 Release-Specific Information for RTI Data Distribution Service 4.5

RTI Data Distribution Service 4.5 is compatible with 4.2 - 4.4, except as noted below.

2.1.2.1 RTPS Versions

RTI Data Distribution Service 4.5 supports RTPS 2.1. Earlier releases (4.2c and lower) supported RTPS 1. Because the two RTPS versions are incompatible with each other, applications built with 4.2e and higher will not interoperate with applications built using 4.2c or lower.

2.1.2.2 double, long long, unsigned long long or long double Wire Compatibility

If your *RTI Data Distribution Service* 4.3 or higher application's data type uses a 'double,' 'long long,' 'unsigned long long,' or 'long double,' it will not interoperate with *RTI Data*

Distribution Service applications built with version 4.2e or lower, unless you use the **-use42eAlignment** flag when generating code with *rtiddsgen*.

2.1.2.3 Sending 'Large Data' between 4.4d and Older Releases

The 'large data' format in *RTI Data Distribution Service* 4.2e, 4.3, 4.4b and 4.4c is not compliant with RTPS 2.1. 'Large data' refers to data that cannot be sent as a single packet by the transport.

This issue has been resolved, starting with *RTI Data Distribution Service* 4.4d. As a result, by default, large data in *RTI Data Distribution Service* 4.4d and higher is not compatible with older versions. You can achieve backward compatibility by setting the following properties to "1".

```
dds.data_writer.protocol.use_43_large_data_format
dds.data_reader.protocol.use_43_large_data_format
```

The properties can be set per *DataWriter/DataReader* or per *DomainParticipant*.

For example:

```
<participant_qos>
    <property>
        <value>
           <element>
               <name>
                   dds.data_writer.protocol.use_43_large_data_format
               </name>
               <value>1</value>
           </element>
           <element>
               <name>
                   dds.data_reader.protocol.use_43_large_data_format
               </name>
               <value>1</value>
           </element>
        </value>
   </participant_qos>
```

2.2 Code Compatibility

2.2.1 General Information (All Releases)

RTI Data Distribution Service uses an API that is an extension of the OMG Data Distribution Service (DDS) standard API. RTI strives to maintain API compatibility between versions, but will conform to changes in the OMG DDS standard.

RTI Data Distribution Service primarily consists of a library and a set of header files. In most cases, upgrading simply requires you to recompile your source using the new header files and link the new libraries. In some cases, minor modifications to your application code might be required; any such changes are noted in this document.

RTI allows you to define the data types that will used to send and receive messages. To create code for a data type, *RTI Data Distribution Service* includes a tool called *rtiddsgen*. For input, *rtiddsgen* takes a data-type description (in IDL, XML, XSD, or WSDL format); *rtiddsgen* generates header files (or a class in Java) that can be used to send and receive data of the defined type. It also generates code that takes care of low-level details such as transforming the data into a machine-independent representation suitable for communication.

While this is not the common case, some upgrades require you to regenerate the code produced by *rtiddsgen*. The regeneration process is very simple; you only need to run the new version of *rtiddsgen* using the original input IDL file. This process will regenerate the header and source files, which can then be compiled along with the rest of your application.

2.2.2 Release-Specific Information for RTI Data Distribution Service 4.5

2.2.2.4 Type Support and Generated Code Compatibility

- **long long Native Data Type Support**

  Starting with release 4.5c, we assume all platforms natively support the 'long long' data type. There is no longer a need to define RTI_CDR_SIZEOF_LONG_LONG to be 8 on some platforms in order to map the DDS 'long long' data type to a native 'long long' type.

- **double, long long and unsigned long long Code Generation**

  If your *RTI Data Distribution Service* 4.3 or higher application's data type uses a 'double,' 'long long,' 'unsigned long long,' or 'long double,' it will not be backwards compatible with *RTI Data Distribution Service* applications built with version 4.2e or lower, unless you use the **-use42eAlignment** flag when generating code with *rtiddsgen*.

- **Changes in Generated Type Support Code**

  The *rtiddsgen*-generated type-support code for user-defined data type changed in 4.5 to facilitate some new features. If you have code that was generated using *rtiddsgen* 4.4 or lower, you must regenerate that code using the version of *rtiddsgen* provided with this release.

❑ **Cross-Language Instance Lookup when Using Keyed Data Types**

In *RTI Data Distribution Service* 4.3, keys were serialized with the incorrect byte order when using the Java and .NET API for the user-defined data type, resulting in incorrect behavior in the **lookup_instance()** and **get_key()** methods when using keyed data-types to communicate between applications in these languages and other programming languages. This issue was resolved in Java starting in version 4.3e rev. 01 and starting in .NET in 4.4b.

As a result of this change, systems using keyed data that incorporate Java or .NET applications using both 4.3 and this release could experience problems in the **lookup_instance()** and **get_key()** methods. If you are affected by this limitation, please contact RTI Support.

This issue does not impact systems using 4.4 and higher versions, or 4.2 and lower versions of *RTI Data Distribution Service*.

❑ **Data Type with Variable Size Keys**

If your data type contains more than one key field and at least one of the key fields except the last one is of variable size (for example, if you use a string followed by a long as the key):

- *RTI Data Distribution Service* 4.3e, 4.4b or 4.4c *DataWriters* may not be compatible with *RTI Data Distribution Service* 4.4d or higher *DataReaders*.
- *RTI Data Distribution Service* 4.3e, 4.4b or 4.4c *DataReaders* may not be compatible with *RTI Data Distribution Service* 4.4d or higher *DataWriters*.

Specifically, all samples will be received in those cases, but you may experience the following problems:

- Samples with the same key may be identified as different instances. (For the case in which the *DataWriter* uses 4.4d or higher, this can happen only if the *DataWriter's* **disable_inline_keyhash** field (in the DataWriterProtocolQosPolicy) is true (this is not the default case).
- Calling **lookup_instance()** on the *DataReader* may return HANDLE_NIL even if the instance exists.

Please note that you probably would have had the same problem with this kind of data type already, even if both your *DataWriter* and *DataReader* were built with 4.3e, 4.4b or 4.4c.

If you are using a C/C++ or Java IDL type that belongs to this data type category in your 4.3e, 4.4b or 4.4c application, you can resolve the backwards compatibility problem by regenerating the code with version of *rtiddsgen* distributed with *RTI Data Distribution Service* 4.4d. You can also upgrade your whole system to this release.

2.2.2.5 Other API and Behavior Changes

- **New on_instance_replaced() method on DataWriterListener**

  Starting with release 4.5c, there is a new DataWriterListener method, **on_instance_replaced()**, which supports the new instance replacement feature. This method provides notification that the maximum instances have been used and need to be replaced. If you are using a DataWriterListener from an older release, you may need to add this new method to your listener.

- **Counts in Cache Status and Protocol Status changed from Long to Long Long**

  Starting with release 4.5c, all the 'count' data types in DataReaderCacheStatus, DataReaderProtocolStatus, DataWriterCacheStatus and DataWriterProtocolStatus have been changed from 'long' to'long long' in the C, C++ and .NET APIs in order to report the correct value for DDS applications that run for very long periods of time. If you have an application written with a previous release that is accessing those fields, data-type changes may be necessary.

- **Changes in RtpsReliableWriterProtocol_t**

  Starting in release 4.4c, two fields in DDS_RtpsReliableWriterProtocol_t have been renamed:

  - Old name: disable_positive_acks_decrease_sample_keep_duration_**scaler**
    New name:disable_positive_acks_decrease_sample_keep_duration_**factor**
  - Old name: disable_positive_acks_increase_sample_keep_duration_**scaler**
    New name: disable_positive_acks_increase_sample_keep_duration_**factor**

  In releases prior to 4.4c, the NACK-only feature was not supported on platforms without floating-point support. Older versions of *RTI Data Distribution Service* will not run on these platforms because floats and doubles are used in the implementation of the NACK-only feature. In releases 4.4c and above, the NACK-only feature has been reimplemented to use fixed-point arithmetic and the new DDS_Long "factor" fields noted above, which replace the DDS_Double "scaler" fields.

❑ **Tolerance for Destination-Ordering by Source-Timestamp**

Starting with release 4.4b, by default, *RTI Data Distribution Service* is less restrictive (compared to older releases) on the writer side with regards to timestamps between consecutive samples: if the timestamp of the current sample is less than the timestamp of the previous sample by a small tolerance amount, **write()** will succeed.

If you are upgrading from *RTI Data Distribution Service* 4.4a or lower, and the application you are upgrading relied on the middleware to reject timestamps that 'went backwards' on the writer side (that is, when a sample's timestamp was earlier than the previous sample's), there are two ways to keep the previous, more restrictive behavior:

- If your DestinationOrderQosPolicy's **kind** is BY_SOURCE_TIMESTAMP: set the new field in the DestinationOrderQosPolicy, **source_timestamp_tolerance**, to 0.
- If your DestinationOrderQosPolicy's **kind** is BY_RECEPTION_TIMESTAMP on the writer side, consider changing it to BY_SOURCE_TIMESTAMP instead and setting **source_timestamp_tolerance** to 0. However, this may not be desirable if you had a particular reason for using BY_RECEPTION_TIMESTAMP (perhaps because you did not want to match readers with BY_SOURCE_TIMESTAMP). If you need to keep the BY_RECEPTION_TIMESTAMP setting, there is no QoS setting that will give you the exact same behavior on the writer side as the previous release.

Starting with release 4.4b, by default, *RTI Data Distribution Service* is more restrictive (compared to older releases) on the reader side with regards to source and reception timestamps of a sample if DestinationOrderQosPolicy **kind** is set to BY_SOURCE_TIMESTAMP: if the reception timestamp of the sample is less than the source timestamp by more than the tolerance amount, the sample will be rejected.

If you are upgrading from *RTI Data Distribution Service* 4.4a or lower, your reader is using BY_SOURCE_TIMESTAMP, and you need the previous less restrictive behavior, set **source_timestamp_tolerance** to infinite on the reader side.

❑ **New Location and Name for Default XML QoS Profiles File (formerly NDDS_QOS_PROFILES.xml)**

Starting with release 4.4d, the default XML QoS Profiles file has been renamed and is installed in a new directory:

- Old location/name:
  $NDDSHOME/resource/xml/NDDS_QOS_PROFILES.xml
- New location/name:
  $NDDSHOME/resource/qos_profiles_<version>/xml/
  NDDS_QOS_PROFILES.example.xml (where <version can be 4.4d, for example)

If you want to use this QoS profile, you need to set up your **NDDSHOME** environment variable at run time, and rename the file **NDDS_QOS_PROFILES.example.xml** to **NDDS_QOS_PROFILES.xml** (i.e., by default, even if your **NDDSHOME** environment variable is set, this QoS profile is not used.) See Section 15.2 in the *User's Manual* for details.

❑ **Changes in the default value for the max_objects_per_thread field**

Starting with release 4.4d, the default value for the **max_objects_per_thread** field in the SystemResourceLimitsQosPolicy has been changed from 512 to 1024.

❑ **Type Change in Constructor for SampleInfoSeq—.NET Only**

Starting with release 4.5c, the constructor for SampleInfoSeq has been changed from SampleInfoSeq(UInt32 maxSamples) to SampleInfoSeq(Int32 maxSamples). This was to make it consistent with other sequences.

❑ **Default Send Window Sizes Changed to Infinite**

- Starting with release 4.5d, the send window size of a *DataWriter* is set to infinite by default. This is done by changing the default values of two fields in DDS_RtpsReliableWriterProtocol_t (**min_send_window_size, max_send_window_size**) to be DDS_LENGTH_UNLIMITED.
- In release 4.4d, the send window feature was introduced and was enabled by default in 4.5c (with **min_send_window_size** = 32, **max_send_window_size** = 256). For *DataWriters* with a HistoryQosPolicy **kind** of KEEP_LAST, enabling the send window could cause writes to block, and possibly fail due to blocking timeout. This blocking behavior changed the expected behavior of applications using default QoS. To preserve that preestablished non-blocking default behavior, the send window size has been changed to be infinite by default starting in release 4.5d.
- Users wanting the performance benefits of a finite send window will now have to configure the send window explicitly.

2.3 Data-Format Compatibility and Extensibility

With *RTI Data Distribution Service*, you can define your own data types, which will be used to communicate between applications on a network.

Developers sometimes extend or modify their data types. When new fields are added, it is a common requirement that applications that use the extended types must still communicate with applications using the old types.

To accomplish this, RTI provides a dynamic data API, which allows applications to define data types at runtime without code generation. Specifically, this API supports the concept of a *sparse type*, one for which every data sample need not contain a value for every field defined in the type. By dynamically defining a type as sparse, an application can add new fields to it later without breaking existing components that may be unable to fill in those new fields.

For customers that may want to handle extensibility in a more custom way, such as with XML payloads or custom serialization and deserialization, RTI provides built-in string and opaque data types.

2.4 ODBC Database Compatibility

To use the Durable Writer History and Durable Reader State features, you must install a relational database, such as Oracle TimesTen or MySQL.

In principle, you can use any database that provides an ODBC driver, since ODBC is a standard. However, not all ODBC databases support the same feature set. Therefore, there is no guarantee that the persistent durability features will work with an arbitrary ODBC driver.

Starting with version 4.5c, we have tested the following drivers:

- Oracle TimesTen 11.2.1
- MySQL ODBC 5.1

Performance will be better with Oracle TimesTen, because it is an in-memory database.

If you choose MySQL, you will also need the MySQL ODBC 5.1.6 (or higher) driver. For non-Windows platforms, UnixODBC 2.2.12 (or higher) is also required.

The Durable Writer History and Durable Reader State features have been tested with the following architectures:

- **AIX:** p5AIX5.3xlc9.0, 64p5AIX5.3xlc9.0 (AIX architectures only support Oracle TimesTen)

- **Linux:** i86Linux2.6gcc3.4.3, x64Linux2.6gcc3.4.5; i86Linux2.6gcc4.1.1, x64Linux2.6gcc4.1.1; i86Linux2.6gcc4.4.3, x64Linux2.6gcc4.4.3 (gcc4.4.3 is only tested with MySQL)
- **Solaris:** sparcSol2.10gcc3.4.2 and sparc64Sol2.10gcc3.4.2
- **Windows:** i86Win32VS2005, i86Win32VS2008, i86Win32VS2010, and x64Win64VS2010

For more information on database setup, please see the *Addendum for Database Setup* (**RTI_DDS_GettingStarted_DatabaseAddendum.pdf**).

2.5 Build Compatibility

2.5.1 Windows CE C/C++ Build Compatibility

When building C/C++ applications on Windows CE systems, there is an additional dependency on **Iphlpapi.lib** that was not required with *RTI Data Distribution Service* 4.4d or previous releases.

Therefore if you rebuild an existing 4.4d or older project, you will see the following link error for static builds.

```
nddscorez[d].lib (Host.obj): error LNK2019: unresolved external symbol
_GetAdaptersAddresses@20 referenced in function
_RTIOsap.Host_getHWAddress*.
```

To correct this error, you must add the new link dependency **Iphlpapi.lib** to the appropriate Visual Studio project. You can do this manually, or you can delete the project files that were created with 4.4d or older and then run **rtiddsgen -example** to replace them.

3 What's Fixed in 4.5d

This section includes:

- Fixes Related to Reliability (Section 3.1)
- Fixes Related to Durability (Section 3.2)
- Fixes Related to QueryConditions and ReadConditions (Section 3.3)
- Fixes Related to rtiddsgen (Section 3.4)
- Fixes Related to TCP Transport (Section 3.5)
- Fixes Related to QoS and Status (Section 3.6)

- ❏ Fixes Related to Dynamic Data in Java API (Section 3.7)
- ❏ Fixes Related to ContentFilteredTopics (Section 3.8)
- ❏ Other Fixes (Section 3.9)

3.1 Fixes Related to Reliability

3.1.1 Unnecessary Reliability Protocol Traffic from Reliable Reader if Writer Never Wrote Samples

A reliable *DataReader* will periodically send reliability-protocol traffic until it receives confirmation from the *DataWriter* that it realizes the *DataReader* has received all the available samples. In the previous release, if the *DataWriter* never wrote any user-data samples, a reliable *DataReader* continued to send periodic reliability-protocol traffic until the *DataWriter* wrote its first user-data sample (even though the *DataWriter* indicated there was no data available before the first user-data sample was written).

This issue has been resolved.

[RTI Bug # 13180]

3.1.2 Failure when Using a Reliable Asynchronous Publisher with Disabled Positive ACKs

A reliable asynchronous-publishing *DataWriter* with **DDS_DataWriterProtocolQosPolicy.disable_positive_acks** set to true would crash when asynchronously writing a sample. This problem has been resolved.

[RTI Bug # 13523]

3.1.3 Dynamic Send Window Stuck at Minimum Size

If a reliable *DataWriter* was configured to have a dynamic send-window size[1], the send window size became stuck at the minimum size and did not grow dynamically. This caused the writer to block due to an unnecessarily small send window and throttled the sending of user data. This problem has been resolved.

[RTI Bug # 13588]

3.1.4 Piggyback Heartbeats Not Always Sent When Send Window Enabled

If a *DataWriter* used a finite send window size with an infinite writer queue size (that is, its **resource_limits.max_samples** was DDS_LENGTH_UNLIMITED or its **writer_resource_limits.max_batches** was DDS_LENGTH_UNLIMITED when batching was enabled), piggyback heartbeat messages were not sent when the *DataWriter's*

1. A 'dynamic send window size' means min_send_window_size was not equal to max_send_window_size.

protocol.rtps_reliable_writer.heartbeats_per_max_samples was greater than its **protocol.rtps_reliable_writer.max_send_window_size**.

This problem has been resolved.

[RTI Bug #13596]

3.1.5 RELIABLE_READER_ACTIVITY_CHANGED StatusCondition Never Reset

When a StatusCondition with the enabled status DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS was attached to a WaitSet, the WaitSet would only block the first time **wait()** was called. It would never block again because the status condition was never reset to false.

This problem has been resolved. The fix ensures that after calling **get_reliable_reader_activity_changed_status()**, this status is reset and the next call to **wait()** will block as expected.

[RTI Bug # 13643]

3.1.6 Undocumented Behavior That Finite Send Window Caused KEEP_LAST DataWriters to Block and Potentilaly Fail on Write

Previously, in 4.4d, the send window feature was introduced to improve throughput. One side effect though was not documented, and its impact could have been detrimental: when a *DataWriter* had a finite send window size and a HistoryQosPolicy kind of KEEP_LAST, its writes may have blocked when its queue was full. Furthermore, the write may have failed when the blocking timeout expired. The blocking with KEEP_LAST deviated from expected behavior significantly.

To resolve this, a *DataWriter* with both a finite send window and KEEP_LAST history may still block on write but it will not timeout; rather, once its max blocking time has elapsed, it will exercise its KEEP_LAST semantics to purge a historical sample from its queue.

Note that the previous default QoS with a finite send window has been changed so that this issue will not be encountered when using default QoS.

3.2 Fixes Related to Durability

3.2.1 Lost Samples in Late-Joining Reader when using Keyed Topic and KEEP_LAST History

Late-joining *DataReaders* may have lost samples coming from *DataWriters* configured with durable writer history. This problem only occurred when using keyed Topics and the *DataWriter's* History QoS policy was configured for KEEP_LAST behavior. This problem has been resolved.

[RTI Bug # 13549]

3.2.2 Slow Restore Operation when using Keyed Topics and dds.data_writer.history.odbc_plugin.in_memory_state Set to 1

The durable writer history 'restore' operation (set with the property **dds.data_writer.history.odbc_plugin.restore**) ran slower when using a keyed Topic and **dds.data_writer.history.odbc_plugin.in_memory_state** set to true. This problem has been resolved.

[RTI Bug # 13487]

3.3 Fixes Related to QueryConditions and ReadConditions

3.3.1 Loss of StateMask Values During QueryCondition's set_query_parameters()

Updating a QueryCondition's search parameters (**set_query_parameters()**) caused the QueryCondition's InstanceState, ViewState, and SampleState masks to be reset to ANY. This problem has been resolved, keeping the original values intact.

[RTI Bug # 13530]

3.4 Fixes Related to rtiddsgen

3.4.1 Error Parsing XML Unions with Enums as Discriminators—C API Only

The XML type parser reported an error when it encountered a union with an enum as the discriminator, such as the following:

```
<union name="MyUnion2" topLevel="1">
    <discriminator type="nonBasic"
     nonBasicTypeName="MyModule::MyEnum"/>
    <case>
        <caseDiscriminator value="MyModule::Enumerator0"/>
        <member name="m1" type="long"/>
    </case>
    <case>
        <caseDiscriminator value="MyModule::Enumerator1"/>
        <caseDiscriminator value="3"/>
        <member name="m2" arrayDimensions="1,2" type="nonBasic"
             nonBasicTypeName="MyModule1::MyModule1Struct1"/>
    </case>
    <case>
        <caseDiscriminator value="default"/>
        <member name="m3" type="float"/>
    </case>
</union>
```

This problem has been resolved.

[RTI Bug # 13176]

3.4.2 Some Constants Generated with -convertToXML were not Language-Neutral

When generating XML files with **-convertToXml**, some constants were not translated to XML in a language-neutral manner.

For example, the following IDL resulted in incorrect XML that used the C representation of TRUE and FALSE:

```
//Union with boolean
union BooleanUnion switch (boolean){
    case TRUE:
        short sData;
    case FALSE:
        long lData;
}; //@top-level true
```

[RTI Bug # 13472]

3.4.3 Use of Keyword 'any' in IDL Resulted in Code that Did Not Compile

When the keyword 'any' is used in an IDL file, *rtiddsgen* processed the file without reporting an error or warning, but then generated code that did not compile. This problem has been resolved. Now *rtiddsgen* will print an error message if 'any' is encountered in the IDL and it will not generate code.

[RTI Bug # 13496]

3.4.4 USER_QOS_PROFILE.XML Not Generated for Mac OS X Platform

rtiddsgen did not create a **USER_QOS_PROFILE.xml** file when it was used to create example code for the Mac OS X platform (x64Darwin10gcc4.2.1). This problem has been resolved.

[RTI Bug # 13504]

3.4.5 Error when Running rtiddsgen with Java Option and XSD in Another Directory

You may have seen a null pointer exception when running *rtiddsgen* with the Java language option and an XML Schema **.xsd** file not in the current working directory. This problem has been resolved.

[RTI Bug # 13534]

3.4.6 Incorrect Code Generated if XSD Enumerations had Non-Ordinal Annotations

If you used an XML Schema **.xsd** file that had enums with <xs:enumeration> tags that contained annotations other than <xsd:annotation><xsd:appinfo><ordinal>, *rtiddsgen* generated code that would not compile. For example, the following XSD snippet generated incorrect code:

```
<xs:enumeration value="ACCS">
    <xs:annotation>
        <xs:documentation>
            <doc:EnumerationDoc>
            <doc:FullName>Air Command and Control System</doc:FullName>
            </doc:EnumerationDoc>
        </xs:documentation>
    </xs:annotation>
</xs:enumeration>
```

This problem has been resolved.

[RTI Bug # 13537]

3.4.7 Incorrect XML Generated When Using Some Preprocessors

Previously, *rtiddsgen* may have created incorrect XML output (when using the option **-convertToXml**) when using some preprocessors, such as MinGW on a Windows system. The output XML files could not be parsed by tools such as *RTI Routing Service* or *RTI Recorder*. This problem has been resolved.

[RTI Bug # 13541]

3.4.8 Error from rtiddsgen if IDL File Contained Unicode non-ASCII Characters

In the previous release, *rtiddsgen* reported an error if there were Unicode, non-ASCII characters in the IDL file. This problem has been resolved.

[RTI Bug # 13581]

3.4.9 Issues when using rtiddsgen -D Incorrectly

Previously, if you invoked *rtiddsgen* with the **-D** option but failed to specify an IDL file, *rtiddsgen* reported a NullPointerException and crashed. For example:

```
C:>rtiddsgen -D MACRO=long -example x86Win32VS2008
Exception in thread "main" java.lang.NullPointerException
     at com.rti.ndds.nddsgen.Main.processArguments(Main.java:1817)
     at com.rti.ndds.nddsgen.Main.main(Main.java:784)
```

This problem has been resolved. Now *rtiddsgen* will report the syntax error.

[RTI Bug # 13604]

3.4.10 Wrong Visual Studio Properties for Hello_idl Example and Generated Examples

When using Visual Studio to compile example code for Hello_idl, you may have seen error messages similar to the following :

```
error MSB3073: The command "copy
C:\ProgramFiles\RTI\ndds.4.5c\lib\i86Win32dotnet2.0\*.dll C:\Program
Files\RTI\helloworlddotnet40\bin\Debug\:VCEnd" exited with code 1.
C:\Program
Files\MSBuild\Microsoft.Cpp\v4.0\Microsoft.CppCommon.targets 113 6
HelloWorld_type
```

These errors only occurred if the installation path (NDDSHOME) included spaces (such as "c:\Program Files").

This problem has been resolved.

[RTI Bug # 13641]

3.4.11 Error Parsing XML Typedefs with Non-Literal Dimensions

The XML type parser failed if the file contained typedefs of sequences, strings or arrays with non-literal dimensions (that is, they referred to a constant).

For example, this worked:

```
<typedef name="Typedef1" sequenceMaxLength="40" type="long"/>
```

But this resulted in an error:

```
<const name="CONST1" type="short" value="40"/>
<typedef name="Typedef1" sequenceMaxLength="CONST1" type="long"/>
```

The error messages were:

```
DDS_XMLTypeCodeParser_parse_from_file:Error parsing XML
DDS_XMLTypeCode_get_const_value:failed to retrieve type-code kind
```

This problem has been resolved.

[RTI Bug # 13687]

3.4.12 Incorrect Const Enums when Using rtiddsgen -package option

When using *rtiddsgen* with the options "-language Java -package test" IDL which specifies a constant enumeration was not processed correctly.

For example, consider the following IDL file:

```
ConstEnumProblem.idl:
module first {
   enum TestEnum { a, b, c };
   const first::TestEnum FIRST_CONST_ENUM = first::TestEnum::c;
```

```
};

module second {
   const first::TestEnum SECOND_CONST_ENUM = first::TestEnum::c;
};
```

The above IDL resulted in the following generated code:

```
FIRST_CONST_ENUM.java:
public class FIRST_CONST_ENUM {
   public static final test.first.TestEnum VALUE = (first.TestEnum.c);
}
SECOND_CONST_ENUM.java:
public class SECOND_CONST_ENUM {
   public static final test.first.TestEnum VALUE = (first.TestEnum.c);
}
```

In both cases, the value of the enumeration was missing the package "test" that is necessary to resolve the enumerate. Note that the data type correctly resolved to test.first.TestEnum.

[RTI Bug # 13719]

3.4.13 Error Running rtiddsgen for Some Linux Distributions

Running *rtiddsgen* for some Linux architectures (such as 64-bit Ubuntu systems) failed with the following error:

```
/opt/RTI/ndds.4.5c/scripts/rtiddsgen: JREHOME not set.
```

This problem has been resolved.

[RTI Bug # 13722]

3.4.14 Incorrect Type Code Name when Using -package

When using *rtiddsgen* with the **-package** option, the typecode included the package name but the typename did not. This inconsistency has been resolved. The package name will no longer be included in the typecode.

[RTI Bug # 12550]

3.4.15 Incorrect Case Values for Unions with Enum Discriminators

Unions with enumerator discriminators were not evaluated correctly. In the following code, the case value ONE should evaluate to 1. However in the previous release, the generated code used the value 3 (the ordinal value of ONE in Enum1).

```
union Enum1 {
    ONE = 3, TWO = 2, THREE = 1
};
```

```
union Enum2 {
    ONE = 1, TWO = 2, THREE = 3
};
union Union1 switch(Enum2){
    case ONE: long m1;
    default: long m2;
};
```

This problem has been resolved.

[RTI Bug #13783]

3.4.16 Compiler Error if IDL Union had Field Named 'result' or 'value'—Java API Only

If an IDL union was declared with a non-primitive field named **result**, this caused a name conflict in the generated Java code and it would not compile. This only happened when the field's **hashCode()** method was called, since the generated code used a local variable called **result**. A similar name conflict could have also occurred if the IDL used name **value** with the **result()** and **other()** methods. This problem has been resolved.

[RTI Bug #13780]

3.4.17 Inconsistent Header Files Created When Generating Code for C or C++

Running rtiddsgen twice on the same IDL file did not produce identical results. The type-specific header file generated by *rtiddsgen* (such as MyType.h) was not identical for each execution of *rtiddsgen*. This problem has been resolved.

[RTI Bug #13809]

3.5 Fixes Related to TCP Transport

3.5.1 TCP Transport Plugin Aborted During Participant Restart

When running TCP LAN in symmetric mode and restarting a participant, the TCP transport plugin would stop working and report the error message "Control thread is stopped because of an internal critical error." This problem has been resolved.

[RTI Bug # 13552]

3.5.2 Possible Crash after Error in TCP Transport Protocol

The TCP Transport may have crashed if there was an error in the protocol that manages the transport connections. In this case, you may have seen the following error messages prior the crash:

```
NDDS_Transport_TCP_ControlErrorCode_deserialize:error deserializing
CONTROL attribute
NDDS_Transport_TCP_ControlAttribute_deserialize:error deserializing
```

```
CONTROL ERROR-CODE attribute
NDDS_Transport_TCP_ControlMessage_deserialize:error deserializing CON-
TROL SERVER-LOGICAL-PORT-ERROR-RESPONSE message
NDDS_Transport_TCP_ControlMessageManager_get_message_from_buffer:error
deserializing CONTROL message
NDDS_Transport_TCPv4_Plugin_processControlEvent:failed to deserialize
control message
```

[RTI Bug # 13761]

3.5.3 Segmentation Fault if TCP Transport Failed to Create Socket

If the TCP Transport was unable to create a socket, you may have seen a segmentation fault. This problem has been resolved.

[RTI Bug # 13760]

3.6 Fixes Related to QoS and Status

3.6.1 Incorrect uncommitted_sample_count Status Caused '!precondition' Exception or Crash

The **DDS_DataReaderProtocolStatus.uncommitted_sample_count**, introduced in 4.5c, was not updated correctly. When observed, it was most likely set to the invalid value of -1. Furthermore, when using debug libraries, a benign precondition exception would be logged due to this status. When using release libraries, it was possible (but rare) for the reader to crash. This problem has been resolved.

[RTI Bug # 13546]

3.6.2 Use of cpu_list in Thread Settings May Have Caused Memory Corruption

When using the **cpu_list** field in ThreadSettings_t, there was a potential memory corruption when the DomainParticipant's QoS was freed, went out of scope, or was finalized.

The DomainParticipant's Event, Database, and ReceiverPool QoS policies all have a ThreadSettings_t field (named thread), which contains a sequence. In the previous release, the middleware performed a shallow copy of DomainParticipantQos, instead of doing a deep copy for those QoS policies.

As a result, it was possible to have multiple DomainParticipant QoS policies pointing to the same memory for the sequences. When one of those QoS policies was finalized, freed, or went out of scope, there was a potential problem in accessing freed memory.

[RTI Bug # 13572]

3.6.3 Some Participant Properties not Propagated, Serialization Error Reported

Not all of a *DomainParticipant's* properties (configured with Property QoS policy) with the propagate flag set to true were propagated, and you may have seen a serialization error. This problem has been resolved.

[RTI Bug # 13589]

3.6.4 Fields in DataWriterProtocolStatus Not Properly Filled In

The following fields in the DDS_DataWriterProtocolStatus structure were not properly filled in:

- **first_available_sample_sequence_number**
- **last_available_sample_sequence_number**
- **first_unacknowledged_sample_sequence_number**
- **first_available_sample_virtual_sequence_number**
- **last_available_sample_virtual_sequence_number**
- **first_unacknowledged_sample_virtual_sequence_number**
- **first_unacknowledged_sample_subscription_handle**
- **first_unelapsed_keep_duration_sample_sequence_number**

This problem has been resolved.

[RTI Bug # 13559]

3.6.5 Partition Matching Errors after Partition is Changed

Changing a *Subscriber's* partition after the *DataReader* was enabled may have caused the *DataReader* to incorrectly receive data from *DataWriters* that were not in the partition. This problem has been resolved.

[RTI Bug # 13612]

3.7 Fixes Related to Dynamic Data in Java API

3.7.1 DynamicData.get_string() Failed When Using Member ID—Java API Only

When using the Java API, calling **DynamicData.get_string()** with a member ID instead of a member name resulted in an error.

For example, this worked:

```
DynamicData.get_string("memberX", DynamicData.MEMBER_ID_UNSPECIFIED);
```

But this resulted in an error:

```
DynamicData.get_string(null, 3);
```

The error messages were:

```
via name: Hello World!
DDS_DynamicData_get_string:!get member name
via id: null
```

This problem has been resolved.

[RTI Bug # 13427]

3.7.2 Setting of Unsigned Types Not Supported with Dynamic Data API—Java API Only

Because Java does not have unsigned types, the Java Dynamic Data API does not provide methods such as **set_ulong()**, but only **set_long()**.

Before this release, if a member "X" was declared as unsigned (e.g., unsigned long X), calling **sample.set_long("X", 10)** would fail.

This release eases the restrictions on calling **set_long()** on a member declared as an unsigned long. Now calls such as **sample.set_long("X", 10)** will succeed even though the type of the underlying IDL member, X, is an unsigned long. This change applies to:

- **DynamicData.set_long()**
- **DynamicData.set_longlong()**
- **DynamicData.set_short()**
- **DynamicData.get_long()**
- **DynamicData.get_longlong()**
- **DynamicData.get_short()**

[RTI Bug # 13536]

3.8 Fixes Related to ContentFilteredTopics

3.8.1 Best-Effort Writer-Side Content-Filtered Samples Reported As Lost

Given a best-effort *DataWriter* with writer-side content-filtering enabled, and with either Liveliness **lease_duration** set to infinite or Liveliness **kind** set to AUTOMATIC_LIVELINESS, the *DateWriter's* matching *DataReaders* could have incorrectly reported content-filtered samples as lost samples. Consequently, the *DataReaders* incorrectly updated the 'sample lost' status in DDS_SampleLostStatus. This problem has been resolved.

[RTI Bug #13476]

3.8.2 Reliable Content-Filtering Writer May Have Incorrectly Inactivated Reader

A reliable *DataWriter* performing writer-side content filtering may not have sent piggyback heartbeats for filtered samples. This would result in the *DataWriter* incorrectly inactivating the corresponding *DataReader*, because the *DataReader* did not have the chance to assert its activity by responding to the piggyback heartbeat. This problem has been resolved.

[RTI Bug #13636]

3.8.3 No Warning if contentfilter_property_max_length was Too Small

When creating a DataReader on a content filtered topic, the middleware did not check whether **contentfilter_property_max_length** (in the DomainParticipantResourceLimitsQosPolicy) was consistent.

If **contentfilter_property_max_length** was not big enough, no warning printed and no information related to the content filter was sent during discovery. However, this did not prevent the DataReader from filtering the data, but it was less efficient.

Now you will see a warning message if this situation occurs.

[RTI Bug #13741]

3.8.4 Improper Serialization of content_filter_property for SubscriptionBuiltinTopicData

When serializing content_filter_property, as part of DDS_SubscriptionBuiltinTopicData, the serialization process did not account for the serialization overhead properly. The algorithm has been corrected.

[RTI Bug #13741]

3.8.5 Failure When Immediately Resending Writer-side Content-Filtered Sample

Previously, when a reliable writer resent a writer-side content-filtered sample in response to a negative acknowledgement, if the resend was done with zero response delay, the sample might not have been serialized properly.

Consequently, the reader would drop the sample with the following exception:

```
PRESCstReaderCollator_storeInlineQos:!dropping message due to incom-
patible inline QoS
PRESCstReaderCollator_storeSampleToEntry:!store inline qos in entry
```

You may have worked around this issue by disabling writer-side content-filtering. This problem has been resolved.

[RTI Bug # 13789]

3.9 Other Fixes

3.9.1 Dispose Samples not Generated for Remote Participant's Contained Entities After Ungraceful Termination

If a DDS application exited ungracefully (that is, without explicitly deleting the participants), other applications listening to discovery traffic would eventually generate a 'dispose participant' sample. However dispose samples were not generated for the participant's *DataReaders/DataWriters*. Therefore, applications that relied on discovery events did not learn that the remote *DataReaders/DataWriters* of other applications that exited ungracefully were no longer alive. This problem has been resolved.

[RTI Bug # 13328]

3.9.2 Error when Using Separate Threads to Enable Participant and Create DataReader at the Same Time

In the previous release, it was not safe for one thread to call **DDS_DomainParticipant_enable()** while another thread called **create_datareader()** or **get_builtin_subscriber()** (which in turn calls **create_datareader()**). This issue has been resolved.

[RTI Bug # 13606]

3.9.3 Wrong Behavior for Null Listener with Non-Null Mask—Java API Only

When using the Java API, the middleware threw an exception if you installed a null listener callback on a specific status. For example:

```
subscriber.set_listener(null, StatusKind.DATA_ON_READERS_STATUS);
```

This problem has been resolved.

[RTI Bug # 13624]

3.9.4 DATA_ON_READERS Status not Reset on Call to take()

A StatusCondition attached to the ON_DATA_ON_READERS status was not reset when **take()** was called. This problem has been resolved.

[RTI Bug # 13624]

3.9.5 Problem Redirecting Logging to File During DomainParticipantFactory Deletion

When redirecting DDS logging to a file, some debug messages during the deletion of the DomainParticipantFactory were mistakenly sent to the standard output. This problem has been resolved.

[RTI Bug # 13680]

3.9.6 Memory In Use After DDS Application Exited—C++ API Only

In the previous release when using the C++ API, you may have seen "memory in use" error messages reported by memory checking tools, even if you had deleted all the DDS entities and called **DDSDomainParticipantFactory::finalize_instance()**. This problem has been resolved.

[RTI Bug # 12906]

3.9.7 Reader Blocked Indefinitely when Writers Sent Large Data

In the previous release, *DataReaders* sometimes blocked indefinitely when *DataWriters* sent large data packets. This problem has been resolved.

[RTI Bug # 13696]

3.9.8 Multichannel DataWriter did not Use Multicast if DataReader had not Defined a Content Filter

Multi-channel *DataWriters* sent data using unicast instead of multicast if the *DataReader* did not specify a ContentFilteredTopic. This problem has been resolved; multi-channel *DataWriters* will use multicast even if the *DataReader* does not have a ContentFiltered-Topic.

[RTI Bug # 13717]

3.9.9 Possible Crash when Configuring DataWriter to Use Asynchronous Publisher

A *DataWriter* configured to use an asynchronous publisher may have crashed. This problem has been resolved.

[RTI Bug # 13738]

3.9.10 Error Message if Type Code Max. Serialized Length > 65532

If **type_code_max_serialized_length** was greater than 65532, you may have seen the following benign message while creating *DomainParticipant* objects:

```
RTICdrTypeCode_initialize_stream:!precondition: length < (sizeof
(RTICdrLong) + sizeof (RTICdrUnsignedShort))
```

This problem has been resolved.

[RTI Bug #13723]

3.9.11 DataReaderProtocolStatus.first_unacknowledged_sample_subscription_handle not Always Updated

The DataReaderProtocolStatus.first_unacknowledged_sample_subscription_handle status was not updated expediently after new samples were written. Therefore when retrieved, the status may have been an invalid handle. This problem has been resolved.

[RTI Bug #13779]

3.9.12 Crash after Setting Logging Delegate while Monitoring Enabled—.Net API Only

If you set a logging delegate for the DDS Config Logger (by calling **NDDS.ConfigLogger.set_log_delegate(LogCallbackDelegate)**) and monitoring was enabled, the application crashed when the participant was created. This problem has been resolved.

[RTI Bug # 13637]

3.9.13 Visual Studio 6.0 on Windows 2000 was not Supported

The previous release did not work when compiled with Visual Studio 6.0 on a Windows 2000 system. This problem has been resolved.

[RTI Bug 13665]

3.9.14 Sequence-number in Statuses Displayed as Negative Values for Best-effort DataWriters

The sequence numbers in DataWriterProtocolStatus (e.g., **first_available_sequence_number**) are applicable only for reliable writers. For best-effort writers, they were set to SN_UNKNOWN. This caused sequence numbers in statuses to become negative.

This problem has been resolved by setting sequence numbers for best-effort *DataWriters* to 0, so that they are displayed as such in *RTI Monitor*.

[RTI Bug # 13767]

3.9.15 Unnecessary Exception When Monitoring DataWriter with Transient or Persistent Durability

A *DataWriter* with transient or persistent durability currently does not support DataWriterProtocolStatus or DataWriterCacheStatus statuses. Previously, when these statuses were accessed by an application, the application would fail with exceptions. Now, instead of failing, the monitoring application will return warnings about the unsupported statuses.

[RTI Bug # 13678]

4 Known Issues

4.1 Estimate of Message Overhead For Large Data May Be Exceeded

RTI Data Distribution Service underestimates the message overhead of samples with large data payloads (on the order of 64 KB). If not properly compensated by the transport's **message_size_max** property, the send operation will fail.

To work around this issue, **message_size_max** should be sized to account for this overhead. As a rule-of-thumb, 64 KB requires an additional 1024 bytes of overhead.

[RTI Bug # 11995]

4.2 Writer-side Filtering May Cause a Deadline to be Missed

If you are using a ContentFilteredTopic and you set the Deadline QosPolicy, the deadline may be missed due to filtering by a *DataWriter*.

[RTI Bug # 10765]

4.3 Disabled Interfaces on Windows Systems

The creation of a *DomainParticipant* will fail if no interface is enabled *and* the **DiscoveryQosPolicy.multicast_receive_addresses** list (specified either programmatically, or through the **NDDS_DISCOVERY_PEERS** file or environment variable) contains a multicast address.

However, if **NDDS_DISCOVERY_PEERS** only contains unicast addresses, the *DomainParticipant* will be successfully created even if all the interfaces are disabled. The creation of a *DataReader* will fail if its TransportMulticastQosPolicy contains a UDPv4 or UPDv6 multicast address.

4.4 Wrong Error Code After Timeout on write() from Asynchronous Publisher

When using an asynchronous publisher, if **write()** times out, it will mistakenly return DDS_RETCODE_ERROR instead of the correct code, DDS_RETCODE_TIMEOUT.

[RTI Bug # 11362]

4.5 Incorrect Content Filtering for Valuetypes and Sparse Types

Content filters may not filter correctly if (a) the type is a valuetype or sparse type using inheritance and (b) the filters refer to members of a derived class.

This issue exists for Topics using DynamicData type support. It may also affect filtering of valuetypes using the .NET or C APIs, or CORBA-compatible C++ type plugins.

[RTI Bug # 12606]

4.6 Multi-dimensional Arrays of Sequences Not Supported when Generating .NET Code

The .NET code generated by *rtiddsgen* for multi-dimensional arrays of sequences is not correct and will not compile.

For example:

```
struct MyStruct {
    sequence<short, 4> m1[3][2];
};
```

[RTI Bug # 13088]

4.7 License File may not be Found if NDDSHOME Set with Quotation Marks

Some distributions of *RTI Data Distribution Service* require a license. One of the places where *RTI Data Distribution Service* may look for a license is the file **rti_license.dat** in the directory specified by the environment variable NDDSHOME. However, if you have set NDDSHOME using quotation marks to enclose the directory path (such as **"C:\Program Files\RTI\ndds.4.5x"**), the license file will not be found there. Instead, you should set NDDSHOME without using quotation marks (such as **C:\Program Files\RTI\ndds.4.5x**).

For more information on where *RTI Data Distribution Service* may look for a license file, see the *RTI Data Distribution Service Getting Started Guide*.

[RTI Bug #13477]

4.8 Errors when Using Monitoring Library and Long Names/Values in PropertyQosPolicy

If your DDS application links with *RTI Monitoring Library* and the PropertyQosPolicy of a *DataReader*, *DataWriter*, or *DomainParticipant* has a name longer than 127 characters or a value longer than 511 characters, you will see an error similar to the following in your DDS application:

```
PRESWriterHistoryDriver_initializeSample:!serialize
WriterHistoryMemoryPlugin_addEntryToSessions:!initialize sample
WriterHistoryMemoryPlugin_getEntry:!add virtual sample to sessions
WriterHistoryMemoryPlugin_addSample:!get entry
PRESWriterHistoryDriver_addWrite:!add_sample
```

```
PRESPsWriter_write:!collator addWrite
RTIDefaultMonitorParticipantObject_sampleAndPublishWriterDesc:!write
RTIDefaultMonitorPublisher_onEventNotify:!publish writer desc
```

As a work-around, you can set the following monitoring configuration properties to "true" to prevent the PropertyQosPolicy being sent out as part of the monitoring data for certain type of entities:

```
rti.monitor.config.skip_writer_properties
rti.monitor.config.skip_reader_properties
rti.monitor.config.skip_participant_properties
```

[RTI Bug # 13625]

4.9 Changes Required when Using More than 5 Content-Filter Parameters

The **contentfilter_property_max_length** (in the DomainParticipantResourceLimitsQosPolicy) needs to account for the number of characters that each part of the ContentFilterTopic data (TopicName, CFName, CFExpr, params, etc.) plus one null-termination character for each string.

However, if more than five content-filter parameters are specified, the calculation above may not be sufficient. In this case, when specifying the **contentfilter_property_max_length**, you need to account for at least seven more characters for each extra parameter (over the 5th one).

Otherwise, you may see this error when creating the *DataReader*:

```
DISCBuiltin_serializeContentFilterProperty:content filter serializa-
tion error
PRESWriterHistoryDriver_initializeSample:!serialize
WriterHistoryMemoryPlugin_addEntryToSessions:!initialize sample
WriterHistoryMemoryPlugin_getEntry:!add virtual sample to sessions
WriterHistoryMemoryPlugin_addSample:!get entry
PRESWriterHistoryDriver_addWrite:!add_sample
PRESPsWriter_write:!collator addWrite
DISCSimpleEndpointDiscoveryPluginPDFListener_onAfterLocalReaderEnabl
ed:!write
```

[RTI Bug # 13741]

4.10 Issues with Dynamic Data

- The conversion of data by member-access primitives (**get_X()** operations) is limited when converting to types that are not supported on all platforms. This limitation applies when converting to a 64-bit long long type (**get_longlong()** and

get_ulonglong() operations) and a 128-bit long double type (**get_longdouble()**). These methods will always work for data members that are actually of the correct type, but will only support conversion from values that are stored as smaller types on a subset of platforms. Conversion to 64-bit long longs from a 32-bit or smaller integer type is supported on all Windows, Solaris, and Linux architectures, and any additional 64-bit architectures. Conversion to 128-bit long doubles from a float or double is only supported on Solaris SPARC architectures.

[RTI Bug # 12647]

- DynamicData cannot handle a union with a discriminator that is set to a value which is not defined in the type.

  [RTI Bug # 12855]

- DynamicData may have problems resizing variable-size members that are >= 64k in size. In this case, the method (**set_X()** or **unbind_complex_member()**) will fail with the error: "sparsely stored member exceeds 65535 bytes." Note that it is not possible for a member of a sparse type to be >= 64k.

  [RTI Bug # 12897]